Improving Text to Image Prompts Using FLAN-T5

Ethan Adkins
ethan@ucf.edu
University of Central Florida
Orlando, Florida, USA

Abstract

The objective of this paper is to use the FLAN-T5 sequence-to-sequence model to improve poor user input prompts to the Stable Diffusion image generator by automatically adjusting input prompts. The methodology involves a multi-step pipeline where the input is translated by a FLAN-T5 model that is initially fine-tuned on a prompt pair dataset. The prompt/image output is then evaluated by a combination of a BERT score, CLIP score, METEOR, human metrics, and loss. The key findings of this experiment justify the provided method as valid due to the BERT score, CLIP score, and human evaluation metrics showing improvement, however, there is still room to improve these scores further.

Code: https://github.com/EthanAdkins/CAP6640SemesterProject

1 Introduction & Problem Statement

When generating images with a text-to-image program such as Stable Diffusion, users may have issues or a hard time coming up with a prompt that aligns with what they actually are intending. In many cases some basic/raw prompts have a large chance of generating a nonsensical image. This project aims to solve the problem of creating improved and appropriate prompts when generating images in cases where users are unable to do so. In doing this, the goal is to make this process automated and straightforward.

2 Related Work

Several related works exist that attempt to give more power/control to the user while generating images with Stable Diffusion. In Deckers, Peters, and Potthast's work they demonstrated three techniques for solving this [2]. Specifically, this paper had one method that allowed a user to manually visualize and move an image through different embeddings until it appeared how they wanted, a second method that automatically optimized any given image quality metric, and a final method that allows a user to regenerate parts of images to obtain the desired result [2]. While this work allows for finer control over image output, it mainly relies on a user manually tweaking or making adjustments. Figure 1 below shows the UI for how a user may navigate the embedding space. The red dot indicates where the generated embedding is with the representation of the other styles/embeddings existing in a space around it. My project aims to improve prompts with minimal user input so that even a less technologically literate person may benefit.

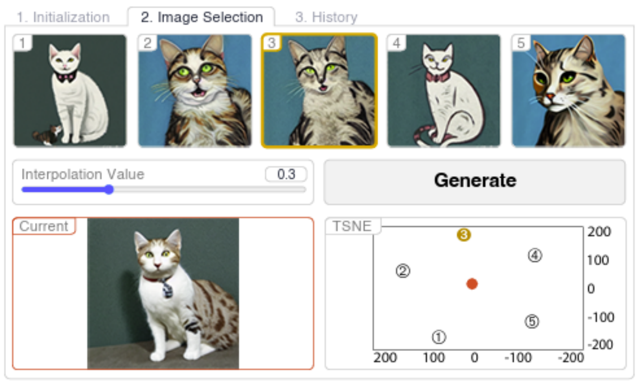


Figure 1: Related Work: User Interface for Human Embedding Adjustment [2]

3 Methodology

The methodology for this project consists of two main components: 1. Training / fine-tuning of the FLAN-T5 model, and 2. Image Generation. The following subsections will first discuss the dataset used as well as a comparison between different versions of T5 and which is optimal for this project. Next will be a visual of the multi-part pipeline used (as seen in Figure 2), followed by a more in-depth discussion of the techniques used in each section.

3.1 Dataset

When training a sequence-to-sequence model it is necessary that the training dataset consists of input/output pair examples. The dataset this project uses is titled "Stable diffusion prompts for instruction models fine-tuning", is based on Hugging Face, and was created by the user 'groloch' [7].

This Stable Diffusion prompt dataset consists of 73.7k training and 8.19k testing prompt pairs. The dataset was compiled/generated by the user scraping prompts people used in the Stable Diffusion Discord chat and using an LLM to convert them from these prompts to a lower/unimproved prompt format (Ex, Detailed->Undetailed). An example of a prompt pair is as follows:

- Initial Prompt: "Winston in Armor, Portrait"
- Improved Prompt: "winston in his armor from overwatch, character portrait, portrait, close up, concept art, intricate details, highly detailed by greg rutkowski, michael whelan and gustave dore"

While this dataset fits the use-case exactly, a lot of the outputs are very similar. What I mean by this is that many of them would have similar styles, artists, or adjectives added to the original prompt. This could be because of who was using the model at the time of the scraping. Therefore, while doing the experimentation, I was cautious about the output being redundant.

3.2 T5

The main model of this project is that of T5. T5 is a sequence-to-sequence model developed by Google that consists of an encoder-decoder architecture [5]. A main goal of this model is to solve the

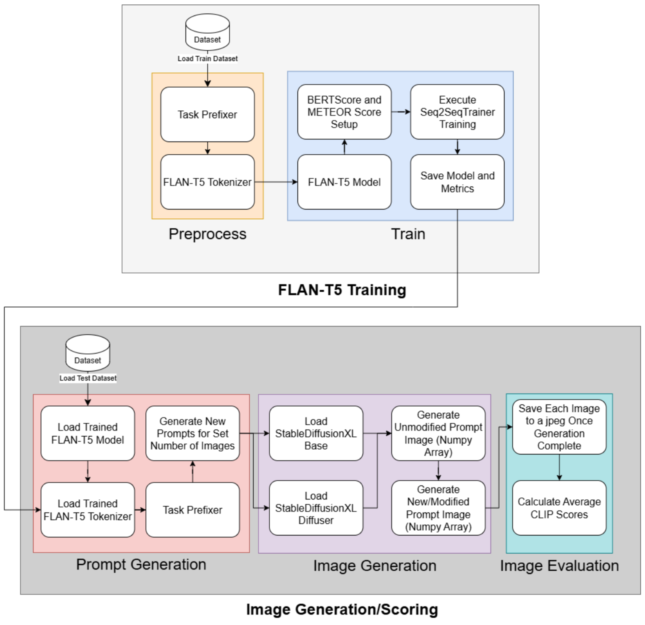

Dataset
Load Train Dataset
Task Prefixer
FLAN-T5 Tokenizer
Preprocess
BERTScore and METEOR Score Setup
Execute Seq2SeqTrainer Training
FLAN-T5 Model
Save Model and Metrics
Train
FLAN-T5 Training
Dataset
Load Test Dataset
Load Trained FLAN-T5 Model
Generate New Prompts for Set Number of Images
Load Trained FLAN-T5 Tokenizer
Task Prefixer
Prompt Generation
Load StableDiffusionXL Base
Load StableDiffusionXL Diffuser
Generate Unmodified Prompt Image (Numpy Array)
Generate New/Modified Prompt Image (Numpy Array)
Image Generation
Save Each Image to a jpeg Once Generation Complete
Calculate Average CLIP Scores
Image Evaluation
Image Generation/Scoring

Figure 2: Overall Pipeline

multi-task issue (similar to what we discussed in class). Its way around this is by converting every natural language processing task into a text generation task [5]. It does this by appending a task prefix to every single input such as "translate", "summarize", etc. Every single input and output are strings of text. Another key note about T5 is that it is already pretrained on several tasks. The original research paper actually discuss pre-training and evaluating on 24 different tasks [10]. Continuing this, the model was specifically pretrained on supervised and unsupervised groups of multi-tasks [10].

3.3 FLAN-T5

Initially, when working on the project, it was decided that the normal T5-base model would be used in order to be fine-tuned for generating the improved prompt pairs. While this implementation was still working and had decent results, I noticed in class that "FLAN-T5" existed and looked more into it. FLAN-T5 is the same exact architecture, parameters, etc as T5, however, it has been fine-tuned on thousands of additional tasks and languages [1]. This additional fine-tuning improves the performance on general tasks as well as few/zero-shot use of the model [1]. Since this project is framing the rephrasing of prompts as a translation problem, as well as the fine-tuning performance increase, the focus of this report will be on the FLAN-T5 model and architecture.

3.4 Pipeline

3.4.1 Training. The training of the model takes place in the "Semester-Project.py" script, which is separate from the Image Generation script (which is "Inference.py"). Each of the scripts were mainly based on the Hugging Face documentation with the appropriate adjustments made (references are in the top of the scripts for what was consulted) [6].

This script consists of several preprocessing steps that corresponds to the yellow/orange square of the FLAN-T5 training section in Figure 2. The previously mentioned dataset is loaded using the Hugging Face datasets library. Each instance in the dataset then has a task prefix appended to the beginning of the text. This is required for the FLAN-T5 model to understand what task to fine-tune on. In the case of this project, the task prefix is "translate basic prompt to detailed longer prompt: " with the goal of having the model learn how to translate these prompts. Once the prefix has been appended to each label the script next loads and uses the FLAN-T5 base tokenizer and tokenizes the entire dataset. At this point the preprocessing is complete.

The script now moves into the actual training process, which is observed in the blue square of the FLAN-T5 training section of Figure 2. We first load the FLAN-T5 Base model and enable gradient checkpointing. The gradient checkpointing allows for use of storage when training the model, as there were memory limitations on the graphics card being used (this will be explained further later in the paper). A data collator is also defined based on the tokenizer and model in order to give some padding to the input so that each input is the same length, by this point, the BERTScore and METEOR score evaluators also will have been loaded from the Hugging Face evaluate library [6]. Finally, training arguments must be set up.

The final hyperparameters for this project are: learning rate of 0.0003, training batch size of 8, evaluation batch size of 4, 5 epochs, quantization disabled, and gradient accumulation steps of 8. At the time of the project milestone, the project was using a learning rate of 0.00005. The reasoning behind this change is that initially, training seemed to be taking a long time and converging relatively slowly (taking tons of epochs). Looking through some of the documentation, Hugging Face recommends the value of 0.0003 for use with most tasks [5]. This, combined with the stability that FLAN-T5 added, led to requiring fewer epochs and much faster learning to obtain similar results as T5.

Initially, the batch sizes were set at double the size (16), and there was no gradient accumulation. However, when attempting to train the model, it would crash and run out of memory on the GPU. Because of this, the batch size was halved and gradient accumulation was introduced. This process accumulates the gradients at different batches and only increases the step of the optimizer after a certain number of batches have been performed [4]. This tricks/simulates the machine into being able to train on higher batch sizes, but requires more time and space.

As mentioned in class, mixed precision training is the use of less precise number formats (in this case FP16) in order to make the model train faster, use less memory, and have similar performance once trained. While the original T5 model supported this quantization, the FLAN-T5 does not. Therefore, this parameter was disabled in training. Finally, 5 epochs were selected. Since the FLAN-T5 model is fine-tuned already on a large number of tasks, the number of epochs of this project's fine-tuning does not need to be as extensive.

Once each of the different components (the model, the dataset, the tokenizer, collator, and metrics) has been initialized, they are then passed into the Seq2SeqTrainer. During training, each step calculates the training loss, gradient norm, and adjusts the learning rate dynamically (depending on momentum). At the end of each

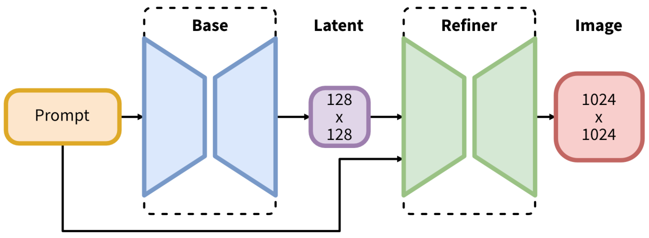


Figure 3: Stable Diffusion XL 1.0 Pipeline [11]

epoch, the previously mentioned metrics are calculated, as well as evaluation loss, BERTScore, and METEOR score. The trainer includes a method for obtaining log history; therefore, all of the information is then logged and saved to a CSV file for future use.

3.4.2 Image Generation & Scoring. After the FLAN-T5 model is trained we are now able to begin generating images. This functionality takes place in the "Inference.py" script, which uses an "ImageEvaluator.py" script for assisting with creating the CLIP scores. The script first verifies that the directories for the images exist and creates them if they do not. Similarly to the training, we now load the dataset, but instead load just the testing set. The tokenizer and model of FLAN-T5 are also loaded. The architecture selects "numImages" number of images from the test set randomly and adds only their first (original) prompts to a list. In this case, we are generating using 100 images/improved prompts, so 100 test prompts are added. For each of these prompts the same task prefix from training is appended to the beginning: "translate basic prompt to detailed longer prompt: ". Finally, for each prompt the trained FLAN-T5 model generates and returns improved prompts that are then decoded by the tokenizer. This project's architecture sets the minimum length of generation to 40 tokens to encourage a longer generation than the input, as 40 is larger than the average unmodified prompt token length. The max length is also set as 1000 to ensure the model does not start hallucinating and/or repeating. Continuing on this thought, a repetition penalty of 1.4 is applied. When researching, it appeared that the creators of this metric believed 1.2 was a good balance between truthful and accurate results without repetition [8]. Unfortunately, the model still repeated some phrases, so the penalty was increased until it seemingly stopped (the 1.4 value).

Stable Diffusion. The model I decided to use for this project is Stable Diffusion's free "Stable Diffusion XL Base 1.0" model and its associated refiner [11]. This model works by generating latent images with a large amount of noise from the base model and feeding this into a refiner to process denoising, as seen in Figure 3 below [11]. It takes as input a prompt, a number of steps, and size of output.

The next steps combines Stable Diffusion with the improved prompts. Each prompt (both the original and improved/generated) is iterated upon and passed to the Stable Diffusion XL base model, which takes 40 generation steps. This outputs in a 512x512 latent image. The latent image is then passed similarly through the refiner, which outputs a numpy array corresponding to the generated image's pixel values. This numpy array is then saved to the computer in storage. Originally the model generated direct images that were not saved and were just continuously loaded in RAM; however, when generating large numbers of images, the virtual RAM of the graphics card being used could not handle this and often crashed. When each of the images are finished being generated, the numpy arrays are then iterated upon and saved as normal images. This is now the moment the images are evaluated and given a CLIP score. Section 4.1.2 discusses more in depth what a CLIP score is but in basic terms it is the semantic similarity between text and an image [3]. In the case of this project, a CLIP score is generated for the similarity between the original prompt and original image, the generated prompt and generated image, and the generated prompt and original image.

3.5 Code Structure and Running

The code for this project is hosted on GitHub in the link directly below the abstract. It consists of four main scripts which will be described.

SemesterProject.py. This file is where the training of the FLAN-T5 model takes place. In order to run, you first must change the "runNum" to indicate where you would like to save the model once trained. Ensure the model name is set to "google/flan-t5-base" as this is the FLAN-T5 model. If you are interested in running the normal T5 model replace this with "google-t5/t5-base". Now to run just execute "python3 .\SemesterProject.py".

Inference.py. This script is responsible for generating the prompt and image pairs based on the trained model from SemesterProject.py. In order to run, you first must navigate to the sd_xl_refiner_1.0 base and refiner models on Hugging Face. A link to both is provided in the README on GitHub. Once downloaded, these files must be placed into the "./CAP6640SemesterProject/StableDiffusionModel" folder for use. It is also important to change "runName" to which run you would like to save the results under and "numImages" to how many image/prompt pairs to generate. You run this by executing "python3 .\Inference.py"

ImageEvaluator.py. This script cannot be run directly and is a helper function to assist with calculating the CLIP scores of the image prompt pairs.

LogHistoryVisualizer.py. This script is used for generating graphs for visualizing the losses, BERTScore, METEOR score, etc. It can be run by doing "python3 ".\LogHistoryVisualizer.py". When running four different windows will open one by one with different graphing of metrics. You may have to validate that the data frame is being assigned a correct/expected path as it references the CSV file that is output by SemesterProject.py.

4 Evaluation & Results

4.1 Metrics

The following three paragraphs will discuss the definitions/functions of each evaluation metric used as well as justification for why they fit in the project's implementation.

4.1.1 BERT Score.

Definition. BERTScore is a metric that uses pre-trained embeddings from the BERT model in order to compare cosine similarity and generate both precision, recall, and F1 scores [12]. Each of these scores ranges from 0 to 1. Rather than just using text overlap, this metric considers the overall semantic meaning of the generated/reference sentences.

Justification. In the case of rewriting/translating a basic input prompt to more complex prompts such as this project is doing a basic overlap metric will act relatively poorly. This is because there is no exact/perfect translation or output, especially in regard to Stable Diffusion prompts. Instead being able to compare the semantic similarity (aka the actual meaning) gives a much better score and performs better for advanced natural language processing evaluation as discussed in lecture. Specifically this score is used and calculated after the training step of the FLAN-T5 model is complete to evaluate the prompt pairs.

4.1.2 *CLIP Score.*

Definition. A CLIP score allows for measurement of how well an image is compatible with a corresponding caption. A number is generated that ranges from 0 to 100 where results closer to 0 imply lower compatibility and numbers closer to 100 imply higher compatibility [3]. The compatibility of the images and captions in other words is really how semantically similar they are.

Justification. This metric is useful for evaluating how well images match the prompt. Due to this, my project calculates the CLIP score after the image generation step. This gives a numerical idea of how closely the image generated by stable diffusion matches the prompt. In this projects case it is important to calculate this score for both the initial prompt / initial image, initial prompt / updated image, and updated prompt / updated image to give an understanding of how well the new prompt both gives better clarity while also retaining the original meaning/context (context wise the previously mentioned BERT Score is also useful in this regard).

4.1.3 *METEOR Score.*

Definition. METEOR is a metric that is specifically used when dealing with machine translation and evaluating the n-gram matching of the input and its translation. Unlike the similar BLEU metric which only considers exact n-gram matches, this metric considers synonyms and alike words.

Justification. Since this project is dealing with a style of translating basic input prompts into more complex ones, adding this metric can complement the BERT score as translation is its main goal. While a low score may not mean much (in regards to negative results) if the model improves the METEOR score this further confirms that the model is working as intended.

4.1.4 *Human Evaluation.*

Definition. Human evaluation is the process of having a person rate or score a model based on their preference, a scoring table, etc. For example, in a text-based model, this can be how fluent the text is that is generated, how coherent it is, and how relevant the output is to the input prompt. In the case of this project, human evaluation will be solely used for evaluating the generated images.

Justification. The process of evaluating the images involves generating both an initial prompt and model improved/rewritten prompt x number of times. For each image, I will designate which one I believe to be the most cohesive image and which matches closest to the original prompt intent. This is important because while numerical metrics may paint a good picture of the improvement the model brings, it may not cover certain images that look poor to the human eye. Many times when a user gives Stable Diffusion a bad prompt, the image will be several solid colors with no meaning or cohesion.

4.1.5 *Loss.*

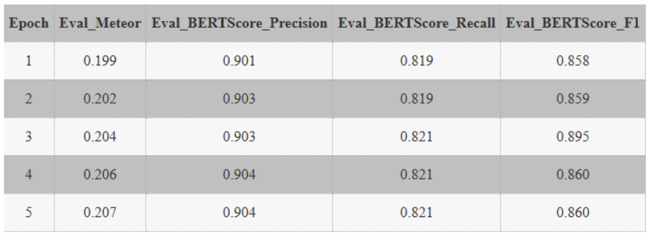
Definition. The broadest definition of loss is a scoring between the difference between what a model generated and what should have been generated. In the case of T5 this loss is the well-known cross-entropy loss [9]. After the T5 tokenizer outputs the original input as a token, the decoder will generate/predict a new sequence and at each step of generation the cross-entropy loss is calculated between the input and prediction [9].

Justification. While every model needs a loss function to learn, it is also important to model the loss visually. This allows for a better understanding of whether the model is learning or getting stuck. If the loss appears to be sporadic, this would indicate poor model parameters. Finally, if the loss is still decreasing when the model finishes training this would indicate further training could improve the model.

4.2 Results

The following sections will now discuss the results and metrics for the prompt generation and image generation results

4.2.1 Prompt Generation Training Results. Figure 4 below displays the METEOR and BERTScore per epoch of training. As expected, the METEOR score is very poor because there is low overlap between the generated/initial prompts, even if they meant the same thing. This issue is resolved with the BERTScore since it uses pretrained embeddings that consider semantic similarity. Each of the three components of this score is very close to human values (with 1 being the highest possible). Though even for the BERTScore metric, the improvement past the first epoch is relatively low. This is most likely due to the zero/few shot learning ability that FLAN-T5 has. When fine-tuning, it likely transferred the learning of the majority of its weights to the new task.



Epoch	Eval_Meteor	Eval_BERTScore_Precision	Eval_BERTScore_Recall	Eval_BERTScore_F1
1	0.199	0.901	0.819	0.858
2	0.202	0.903	0.819	0.859
3	0.204	0.903	0.821	0.895
4	0.206	0.904	0.821	0.860
5	0.207	0.904	0.821	0.860

Figure 4: Evaluation Metric Table (FLAN-T5)

Figure 5 below demonstrates the training and evaluation loss of the training model vs the number of steps. Step 1151 corresponds to epoch 1, and step 5755 corresponds to epoch 5. As time/number of steps increases, the loss decreases in an exponential fashion, as most models do. It does appear, however, that the model may be able to train for another epoch or so while reducing the loss, as the lines are not totally horizontal yet. An interesting observation as well is that the evaluation loss is lower than the training loss. Typically, one would expect the training loss to slightly overfit. Perhaps in the FLAN-T5 model, this is expected.

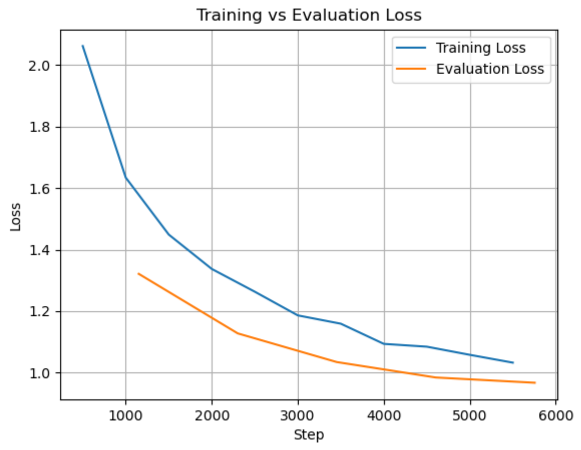


Figure 5: Training Vs. Evaluation Loss

4.2.2 Prompt/Image Generation Results. The following section elaborates on potentially the most important/interesting results, and that is the actual generation step. Below is an example of an initial prompt (from the test set) as well as what was generated by the model output:

- Initial Prompt: "Vintage Mario Art by Masters."
- Generated Prompt: "vintage movie still of mario, intricate, highly detailed, digital painting, artstation, concept art, smooth, sharp focus, illustration, art by artgerm and greg rutkowski and alphonse mucha and william - adolphe bouguereau"

The prompt that is generated adds more complex words and detail, such as "intricate" and "digital painting" while also adding a specific style given by the painters' names at the end. Figure 6 demonstrates what the images for both the initial prompt (left image) and the generated prompt (right image) look like.

A second example is as follows:

- Initial Prompt: "GTA5 Poster: Secret Agent James Conn, Fantasy Style"
- Generated Prompt: "GTA5, poster for secret agent james conn, fantasy, intricate, elegant, highly detailed, digital painting, artstation, concept art, matte, sharp focus, illustration, hearthstone, art by Artgerm and Greg Rutkowski and Alphonse Mucha"

Even though the initial prompt had both a style and the same name for the generated character, the second image in Figure 7 is obviously more coherent. The background of the left/initial image does not have anything to do with the prompt. The right/generated

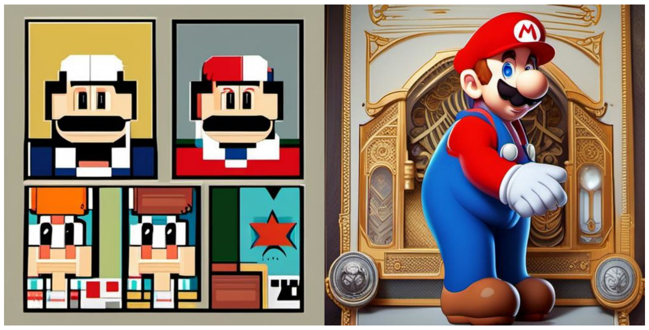


Figure 6: Initial vs Generated Prompt Image Example 1

image does highlight an issue, however. The first prompt had a high reliance on a fantasy style. The model included the fantasy keyword but also added a large number of descriptive words that made the generated image appear to not have much of a fantasy style. This issue is consistent with several image/prompts that are generated by the model.

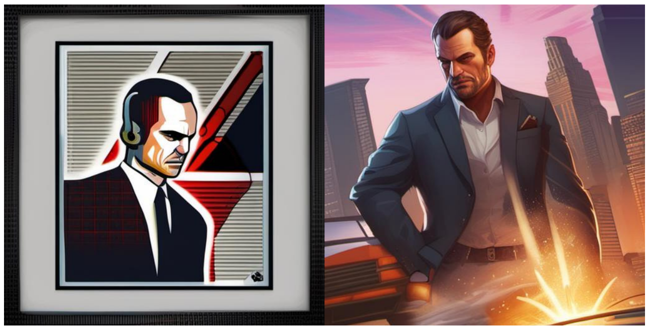


Figure 7: Initial vs Generated Prompt Image Example 2

Word Frequency. I decided to also investigate the frequency/distribution of the most popular words in the 100 original and generated prompts. The top ten words from the original prompts were: "in", "portrait", "style", "by", "art", "a", "with", "of", "artgerm", and "rutkowski". The top 10 most frequent words for the generated dataset were "and", "art", "by", "detailed", "of", "artstation", "concept", "digital", "a", and "painting". The insight gained from this is that the model typically prefers digital-styled art that could be found on art website platforms. The words under the generated prompts also had 5 times as many occurrences. This likely indicates the model is assigning several styles and artists per generated image, which could be throwing off the original prompt's intent. In future work, this could be resolved by reducing the max number of tokens the model can generate.

Human Evaluation Results. Following with section 4.1.4, I went through the 100 randomly generated images and selected which ones I believed to be the most cohesive while also matching the original prompt intent. For example, when evaluating the images in Figure 6, I preferred the second/right image. The left image somewhat resembled Mario in the top right but was overall abstract and incoherent, while the right/generated prompt-based image resolved this issue. In the case where both images were about the same, I opted to choose the original (since there is no need for the generated image). Out of 100 images, I preferred 21 of the original prompts/images and 79 of the generated prompts/images. While there were a small number of cases where the generated prompt led to an incoherent image, there is a much larger chance of this happening with the initial prompts.

CLIP Score Results. The CLIP scores were generated by averaging the score between 100 images and their original and generated prompts. The three different cases are as follows:

- Original Prompt/Original Image CLIP Score: 31.0277
- New Prompt/New Image CLIP Score: 29.0022
- Original Prompt/New Image CLIP Score: 29.4538

When examining the scores together, it appears that in both cases, the generated prompt or the image based on that prompt had slightly lower scores than the original prompt/image pairs. There could be several reasons for this. Since the CLIP score generated is an average of 100 images, there could be a chance that the images that generate poorly with new prompts bring the average down heavily. The original prompt/image will always be similar because the prompt directly generated the image without any changes. Finally, there is also a chance that the model is adding/removing content and context too drastically from the original prompt. This could entirely change the meaning of the initial prompt, leading again to the lower CLIP scores. I will discuss this concept and potentially some solutions in the next section.

5 Discussion & Challenges

The implemented architecture was successful in generating new prompts that expanded upon the original while ensuring the image generated was more coherent, though there are some areas for improvement. An observation made is that while BERTScore demonstrated the generated prompt was semantically similar to the original prompt, the generated image was slightly less similar to the original prompt. In almost every single prompt that was generated, there was an art style and a painter's style added. It may stand to reason that adding these style changes to the original intent too much and while the BERT embeddings/score understood the text was similar, the generated image was further off. An improvement of this could be adding a penalty for adding a style or painter that was not already there, in a similar way to how the repetition penalty works. Another improvement could be using a different metric than the CLIP score for determining similarity. As seen in some of the examples of the results section the images generated do appear to be more coherent and relevant to the human eye. A metric for doing this by using directional similarity, which is another CLIP-style score that can measure the overall consistency between two images [3]. Another insight was that the loss had not plateaued entirely when the 5th epoch finished training. It may be worthwhile in the future to train for twice as many epochs (10) and see if that brings any improvement.

There were several challenges when attempting to create the architecture of this overall project. As previously mentioned, a big challenge was memory and computational power. This was only made worse by the FLAN-T5 model not supporting quantization, meaning either more memory or more time/space was required to train. There were also some challenges when trying to get the related numpy image arrays saved in order to get around these memory issues. The built-in Python sort function sorts alphabetically and does not consider numbers. Since I was labeling each image generated with a number, I had to use a sorting library that had a "natural" sorting function. Another "challenge" is gathering a dataset that has both an input and an output for something such as Stable Diffusion. The dataset I am using requires an LLM to generate the "original" prompts based on the actual prompts. There is no guarantee that the actual prompts that were obtained from the Stable Diffusion Discord were of the best quality they could be. There is even fewer guarantee than an LLM would generate the appropriate "original" or "poor" wording of that prompt.

6 Concluding Remarks

Overall, this paper implemented and contributed an architecture for fine-tuning a FLAN-T5 model with a dataset aimed at improving Stable Diffusion prompts while also generating the associated images with the prompts and evaluating them. The main key takeaways are that the FLAN-T5 model is excellent at transferring its trained tasks to new tasks and can do so relatively quickly, and that the slightest change in style can drastically change the CLIP score (similarity) between an image and its caption for generation. Continuing this, the current model implementation can overwrite or minimize existing styles in the original prompts when attempting to "improve" them. Another takeaway is that providing a dataset of good/bad input prompts to Stable Diffusion allows for a lower number of nonsensical/incoherent image generations.

In the future, it would be worthwhile to train the model for a higher number of epochs with a lower learning rate to see if it can converge even better, while also comparing against different hyperparameters such as variable batch size, etc. Another future work would be comparing/contrasting better evaluation metrics for replacing the CLIP score, or potentially coming up with a new criterion for measuring how well the model performed. The dataset would also benefit from an improvement. This could mean finding a different source of sequence-to-sequence image prompt data that is based on generating images for other prompts and fine-tuning it to Stable Diffusion, artificially generating an entirely new dataset, or looking into other methods. Finally, better hardware and memory would allow for training a future model more quickly and with better results.

References

[1] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. https://doi.org/10.48550/ARXIV.2210.11416

[2] Niklas Deckers, Julia Peters, and Martin Potthast. 2024. Manipulating Embeddings of Stable Diffusion Prompts. arXiv:2308.12059 [cs.CV] https://arxiv.org/abs/2308.12059

[3] Hugging Face. 2024. Evaluating Diffusion Models. https://huggingface.co/docs/diffusers/en/conceptual/evaluation, Last accessed on 2025-4-17.

[4] Hugging Face. 2024. Performing gradient accumulation with Accelerate. https://huggingface.co/docs/accelerate/en/usage_guides/gradient_accumulation, Last accessed on 2025-4-18.

[5] Hugging Face. 2024. T5. https://huggingface.co/docs/transformers/en/model_doc/t5, Last accessed on 2025-4-18.

[6] Hugging Face. 2025. Seq2SeqTrainer. https://huggingface.co/docs/evaluate/transformers_integrations#seq2seqtrainer, Last accessed on 2025-4-18.

[7] groloch. 2024. Stable diffusion prompts for instruction models fine-tuning. https://huggingface.co/datasets/groloch/stable_diffusion_prompts_instruct#stable-diffusion-prompts-for-instruction-models-fine-tuning, Last accessed on 2025-2-17.

[8] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *CoRR* abs/1909.05858 (2019). arXiv:1909.05858 http://arxiv.org/abs/1909.05858

[9] nielsr. 2021. What is loss function for T5. https://discuss.huggingface.co/t/what-is-loss-function-for-t5/11669, Last accessed on 2025-4-21.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[11] stabilityai. 2023. stable-diffusion-xl-base-1.0. https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0, Last accessed on 2025-4-18.

[12] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SkeHuCVFDr